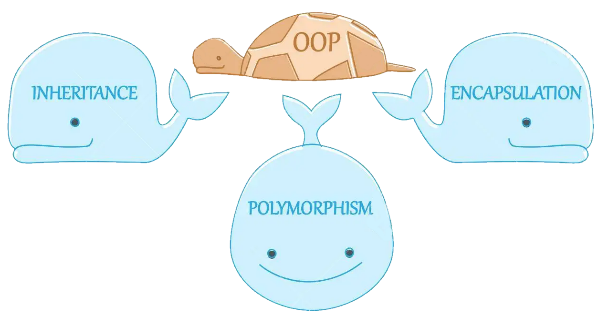
**Lesson 23. Inheritance and Polymorphism**

У даному занятті, ми познайомимось із ще двома китами, на яких стоїть ООП. Одного з них звуть *Наслідування*, а іншого – *Поліморфізм*.

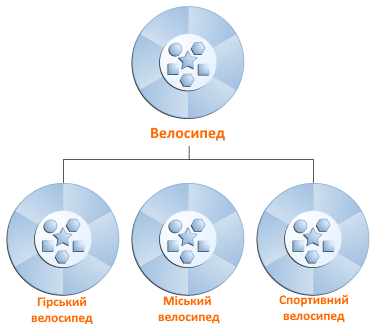
Ми розглянемо їх у комплексі, адже вони тісно між собою пов’язані.

Ці принципи використовуються дуже часто, коли йдеться про використання зовнішніх і не тільки бібліотек Java, що нам дуже знадобиться у подальшій роботі.

**Inheritance**

Отже, **наслідування** – це принцип ОOП, за яким класи можуть переймати (наслідувати) стан і поведінку інших класів.

Ідея полягає в тому, щоб стан і поведінку одного класу можна було повторно використати (а не скопіювати) у іншому класі. Таким чином можна створити клас дуже схожий на інший, але при цьому додати до нього ще щось.



Приклад із реального світу – велосипеди (і ще багато чого іншого). Велосипед (звичайний чи то стандартний, якщо такі досі існують), це пристрій поведінку і стан якого унаслідували дуже схожі пристрої: міський велосипед, спортивний чи гірський.

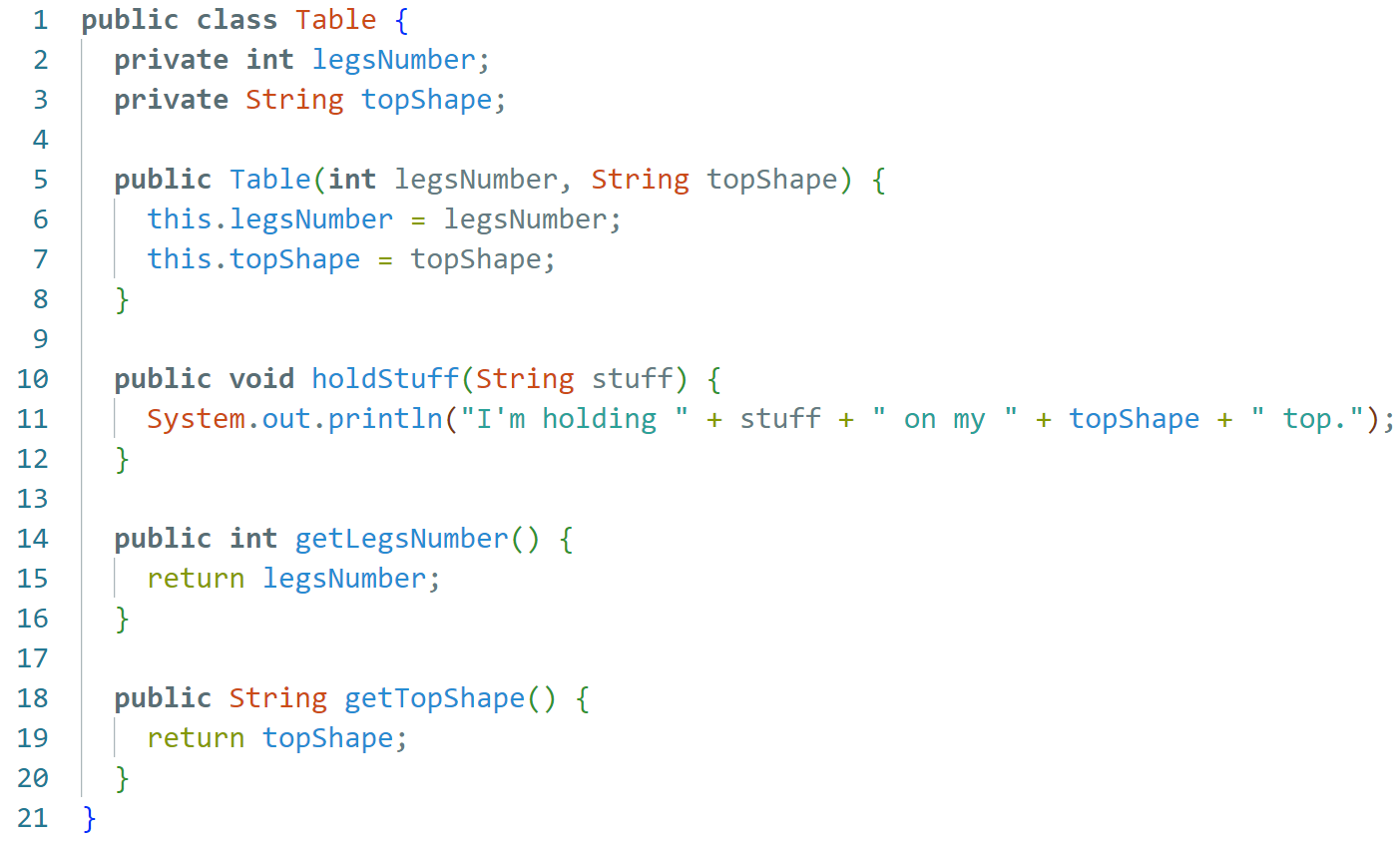
Усі ці три види велосипедів все ще є велосипедами (у всіх є рама, кермо, колеса і на них можна їздити), але кожен із них зазвичай має додатковї функції та частини, які притаманні саме їм.

Аби створити нову модель спеціалізованого велосипеда, не потрібно його знову винаходити і думати, як і куди прилаштувати педалі і гальма, а варто вже відому конструкцію певним чином вдосконалити, що і відповідає дуже добре принципу наслідування.

На прикладі наступного завдання пропонуємо розглянути, як використати принцип наслідування в Java.

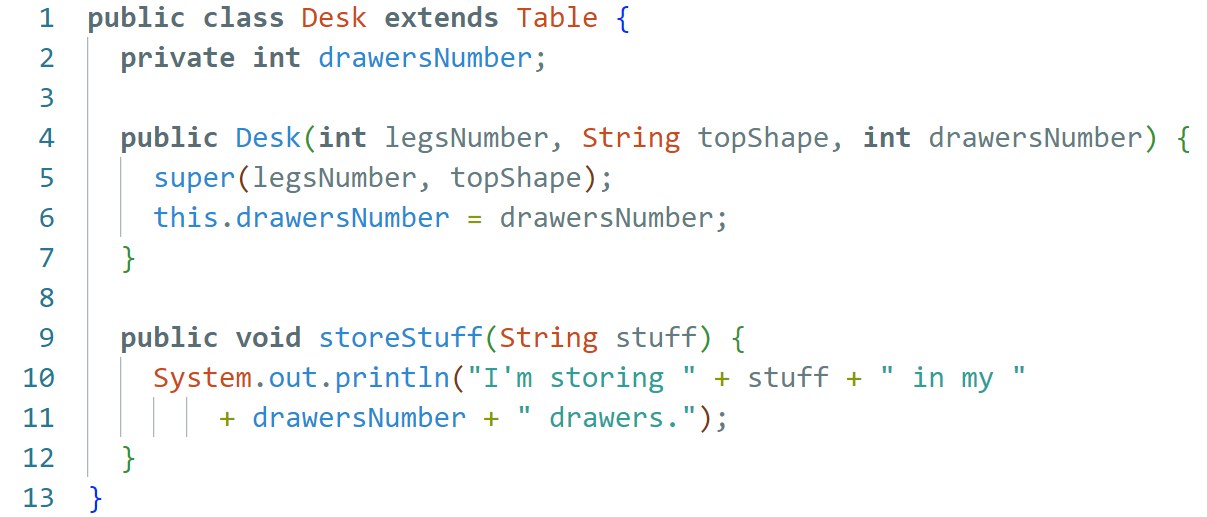
**Example**

**Крок 1**. Для початку розглянемо клас, що відображає модель столу:



Це класичний стіл у якого є ніжки і стільниця певної форми. Він може утримувати речі на собі.

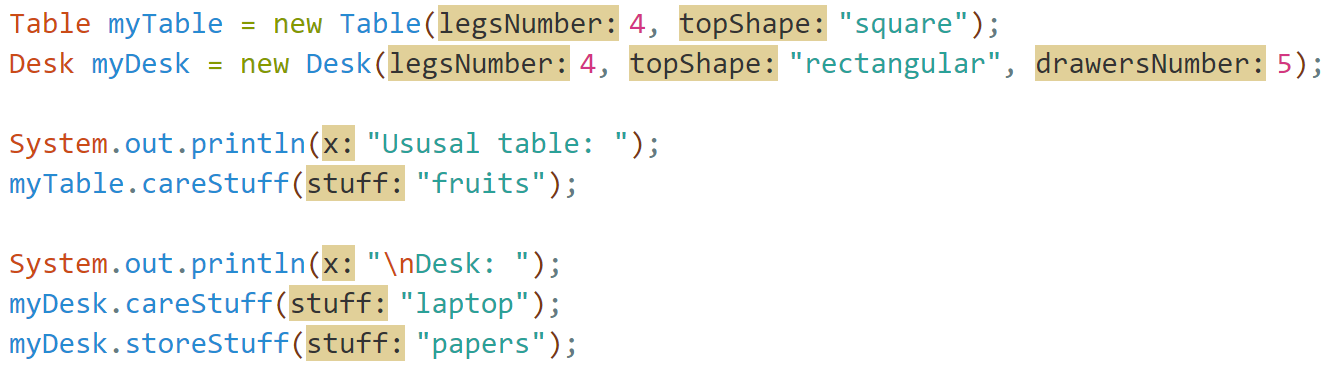
**Крок 2.** Тепер поставимо собі завдання створити клас письмового столу, у якого додатково є шухлядки, де також можна зберігати певні речі. Ось так може виглядати новий клас:



Утворений клас потребує кількох дуже важливих коментарів:

* ключове слово **extends** вказує на клас, з якого поточний клас повинен унаслідувати стан і поведінку. Клас, який наслідує інший клас називають *дочірнім* (або *субкласом*, або *підкласом*), а клас, з якого виконується наслідування – батьківським (або *суперкласом*).
* ключове слово **super** дозволяє звернутись до батьківського класу (його полів та методів). У даному випадку **super**() виконує виклик конструктора батьківського класу. Якщо батьківський клас не має дефолтного конструктора, то в дочірньому конструкторі повинен обов’язково виконуватись виклик конструктора суперкласу.
* такий запис створення класу передбачає, що в ньому вже існують усі поля і методи суперкласу, і доступ до них визначається модифікаторами записаними в батьківському класі.

**Крок 3.** Створимо об’єкти обох класів і перевіримо їх функціональність:



**Polymorphism**

Поліморфізм безпосередньо пов’язаний із наслідуванням. **Поліморфізм** – це ще один принцип ООП, відповідно до якого, дочірній клас може мати відмінну від батьківського класу реалізацію методів.

Тобто класи, які наслідують інший, клас можуть по різному виконувати його методи, а значить їх можна переписати (задати нову поведінку).

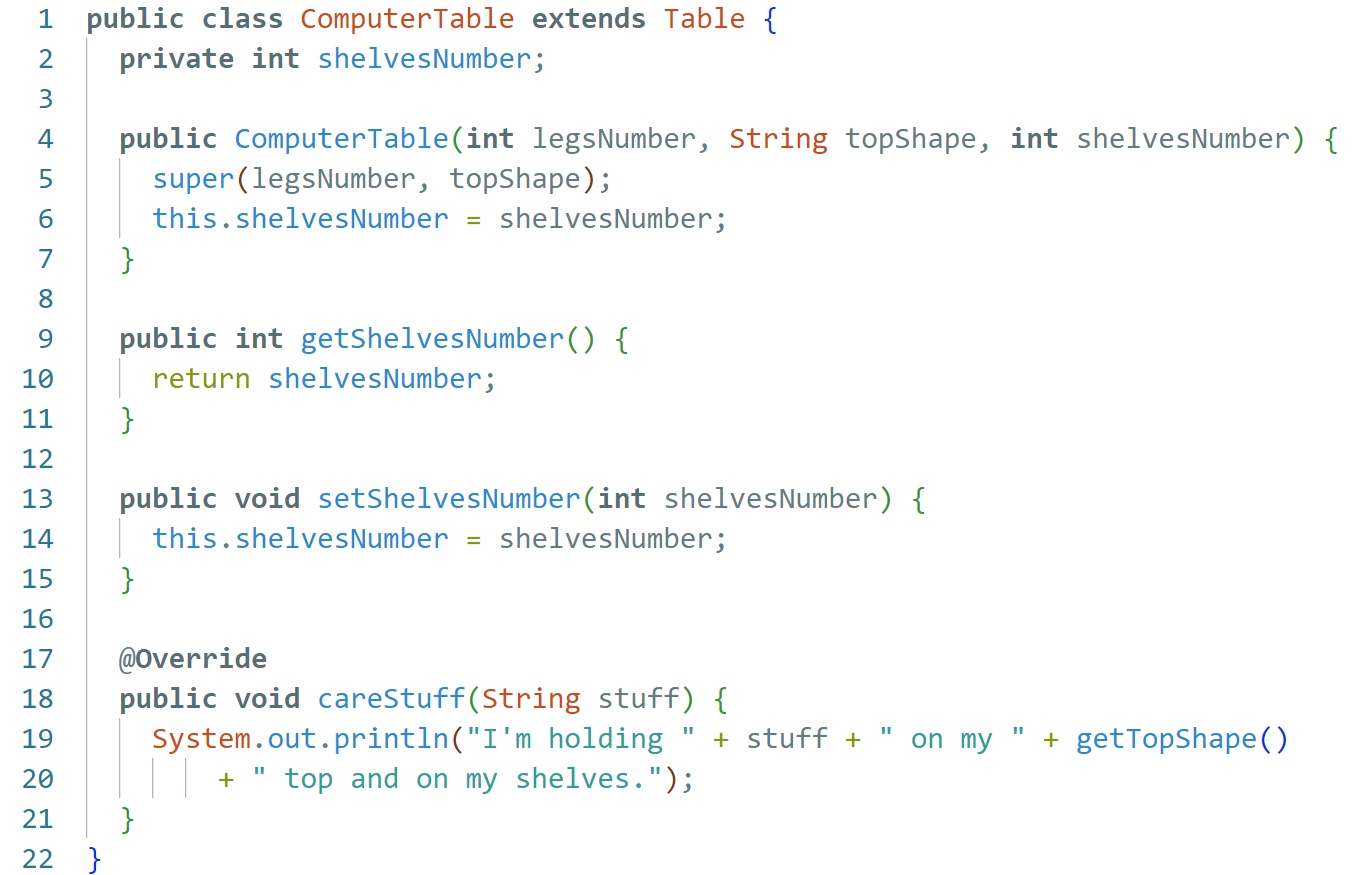
Принцип поліморфізму дозволяє зробити реалізацію наслідування, а отже повторного використання коду, більш гнучкою.

**Example**

Для того щоб дослідити поліморфізм розглянемо наступний приклад. Реалізуємо клас, що відображатиме модель комп’ютерного столу, який наслідуватиме клас Table.

**Крок 1.** Комп’ютерний стіл повинен крім стільниці мати певну кількість поличок, які також будуть утримувати речі на собі. Тобто відповідний метод повинен бути змінений і виконуватись інакше ніж для звичайного столу.

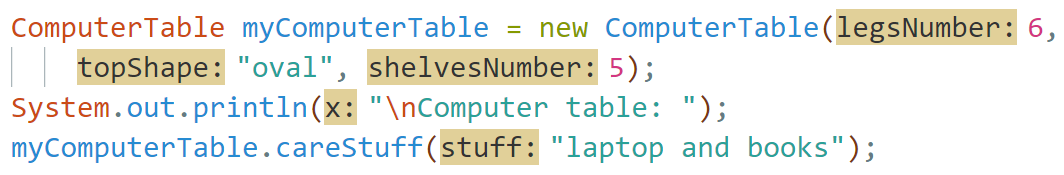
Приклад виконання класу комп’ютерного столу:



**Крок 2.** Даний код також потребує певних коментарів, які додатково розкриють певні правила реалізації поліморфізму:

* для того, що перезаписати один із методів батьківського класу у підкласі, необхідно щоб нове оголошення цього методу мало точно таку ж саму **сигнатуру** (тип, назву, модифікатори та параметри);
* для того, щоб випадково не створити новий метод замість переписування старого перед методом, що повинен бути переписаний вказують декоратор **@Override** (детальніше про це ще поговоримо у наступних заняттях);

**Крок 3.** Створимо об’єкт нового класу і перевіримо його функціонування:



**Homework**

**Завдання 1.** Створіть клас *Person* у якому буде тільки одне поле: повне ім’я (прізвище та ім’я – *fullName*), а серед методів тільки вітання (*sayHello*). При вітанні у консоль повинен виводитись текст на зразок такого: «*John Doe says hello!*».

**Завдання 2.** Створіть клас *Musician* що буде наслідувати *Person*. Об’єкт даного класу також повинен вітатись, але іншою фразою: «*John Doe says hey-hey la-la-la!*». Крім того у музиканта повинна бути змінна, що зберігає інформацію про його музичний існтрумент, та метод *playMusic*, що виводитиме в консоль текст на зразок: «John Doe is playing piano…»